Creando un autocomplete con JavaScript

Felipe Nascimento
28/09/2020

Más conocido como autocomplete, no sólo evitará errores tipográficos, sino que también promoverá otros destinos según lo que el cliente escriba en el campo de búsqueda.

¿Cómo podemos usar **JavaScript** para ayudarnos en esta tarea?

Empezando el autocomplete

Actualmente los nombres de las ciudades en las que la agencia de viajes ofrece los paquetes se almacenan en el siguiente array:

```
const destino = ['Rio de Janeiro', 'San Pablo', 'Nueva York', 'Miami','Roma','
```

Lo primero que haremos es crear una **función** para el autocomplete que recibirá el destino como parámetro y devolverá a los usuarios las opciones.

```
function autoComplete(ciudad) {
        const destino = ['Rio de Janeiro', 'San Pablo', 'Nueva York', 'Miami','
```

```
    return destino;
}
```

Una de las ventajas del autocomplete es escribir una o dos letras y ya recibir algún resultado. Entonces tenemos que encontrar una manera de tomar la cantidad de letras que se escriben y comprobar si hay alguna ciudad que corresponda a estas letras y devolver los resultados al usuario.

Filtrando el array

Para evitar encontrar divergencias entre lo que el usuario escribe y los nombres de las ciudades en array, es necesario estandarizar el input del usuario para comparar con el nombre de la ciudad de una manera más asertiva. Para ello, pondremos todas las letras escritas en minúsculas usando el método [toLowerCase] (https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String/toLowerCase).

Cuando hacemos la comparación, necesitamos devolver una nueva lista, donde sólo los datos que coinciden con lo que el usuario escribió deben estar presentes, para ayudarnos a [filtrar] (https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/filter) iterando por todo el array.

```
    return destino.filter((valor) => {
        const valorMinuscula = valor.toLowerCase()
        const ciudadMinuscula = valor.toLowerCase()
    })
}
```

Ahora que tenemos todas las ciudades en minúscula, para hacer la comparación y devolver la nueva lista en el `filter` usaremos el método (includes) [https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/includes]. Este nos dirá si la string que escribimos, ya sea una letra o el nombre completo de la ciudad, está en la variable `valorMinuscula`. Si la respuesta es cierta, devolvemos esta string al usuario.

```
return valorMinuscula.includes(ciudadMinuscula)
```

Nuestro código completo quedaría así:

```
function autoComplete(ciudad) {
const destino = ['Rio de Janeiro', 'San Pablo', 'Nueva York', 'Miami','Roma','

    return destino.filter((valor) => {
        const valorMinuscula = valor.toLowerCase()
        const ciudadMinuscula = valor.toLowerCase()

            return valorMinuscula.includes(ciudadMinuscula)
        })
}
```

¡Muy bien! El **autocomplete** fue creado, ahora mostraremos el resultado para el usuario =)

Exhibiendo el resultado para el usuario

Tenemos dos campos en nuestro HTML, un campo para que el usuario escriba el destino y una `<ul>`(etiqueta HTML para listas no ordenadas) que vamos a usar para exhibir los resultados:

```
<input class="campo" type="text">
<ul class="sugerencias"></ul>
```

Primero tenemos que capturar los selectores del campo:

```
const campo = document.querySelector('.campo')
const sugerencias = document.querySelector('.sugerencias')
```

Ahora sólo hay que añadir un evento al campo de búsqueda para cuando el usuario digite alguna cosa, el sistema traiga el nombre de las ciudades automáticamente.

```
campo.addEventListener('input', ({ target }) => {
const datosDelCampo = target.value
```

Como la variable `datosDelCampo` está recibiendo lo que se está escribiendo, creamos una condición: caso la string capturada por el evento sea verdadera ponemos el resultado dentro del campo `sugerencias` con la ayuda del innerHTML.

La función `autoComplete`, nos devuelve siempre un array. Necesitamos generar una lista de elementos basada en la lista de sugerencias, para "mapear" la lista de sugerencias para los elementos, vamos a utilizar la función map que será la responsable por iterar los resultados y creará una `<li>` con la respuesta, no olvidando de utilizar el `join` para devolver el resultado como string.

```
if(datosDelCampo.length) {
    const autoCompleteValores = autoComplete(datosDelCampo)
     sugerencias.innerHTML = `
       ${autoCompleteValores.map((value) => {
           return (
                  `<li>${value}</li>`
                )
         }).join('')}
    `}
})
```

Nuestro código con el evento resultó así:

```
function autoComplete(ciudad) {
const destino = ['Rio de Janeiro', 'San Pablo', 'Nueva York', 'Miami','Roma','
    return destino.filter((valor) => {
        const valorMinuscula = valor.toLowerCase()
        const ciudadMinuscula = valor.toLowerCase()
        return valorMinuscula.includes(ciudadMinuscula)
    })
}
const campo = document.querySelector('.campo')
const sugerencias = document.querySelector('.sugerencias')
campo.addEventListener('input', ({ target }) => {
```

```
    const datosDelCampo = target.value
    if(datosDelCampo.length) {
            const autoCompleteValores = autoComplete(datosDelCampo)
            sugerencias.innerHTML = `
                ${autoCompleteValores.map((value) => {
                        return (
                                `<li>${value}</li>`
                            )
                }).join('')}
        `}
})
```

Para saber más

Una otra manera de hacer un rápido autocomplete es a través de las propiedades del HTML5 como [datalist] (https://developer.mozilla.org/es/docs/Web/HTML/Elemento/datalist) que representará un conjunto de elementos y la option value que representa un ítem.

```
<label for="busca">Buscar países</label>
<input type="search" id="busca" list="paises">

<datalist id="destino">
  <option value="Rio de Janeiro"></option>
  <option value="San Pablo"></option>
  <option value="Nueva York"></option>
  <option value="Miami"></option>
  <option value="Roma"></option>
  <option value="Buenos Aires"></option>
  <option value="Madrid"></option>
  <option value="Barcelona"></option>
  <option value="México DF"></option>
  <option value="Bogota"></option>
  <option value="Lima"></option>
```

```
    <option value="Quito"></option>
  </datalist>
```

Si te interesa este tema, aquí en Alura tenemos capacitaciones de front-end, para principiantes y para aquellos que ya son desarrolladores web. En ellas, verás cómo programar en Javascript, HTML y CSS para construir sitios web.

ARTÍCULOS DE TECNOLOGÍA > FRONT END

Acceso a todo el contenido de la plataforma por 6 meses

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 12 meses